SOLENT UNIVERSITY

SCHOOL OF MEDIA ART AND TECHNOLOGY

MSc Computer Engineering

A Master's Thesis in

Smart Traffic Light Controller Towards Smart Cities

September 2022

HASAN ALQAYSI

Supervisor: Dr. Olufemi Isiaq

This report is submitted in fulfilment of the requirements of Solent University for the degree of MSc Computer Engineering.

Abstract

Society must always keep pace with the development of science. Due to the recent advances in surveillance technology, especially in the field of deep learning, the concept of smart cities is closer to be realized than ever. An important aspect of smart cities is smart traffic light control system that can keep up with the enormous increase of vehicles. Also, these systems are necessary as more autonomous cars are produced. The work in this thesis proposes a system that uses deep learning to control traffic light systems to enable smart cities. The system uses deep learning-based object detection in order to detect different types of vehicles on busy intersection. A YOLOv4-tiny model is trained using Darknet framework in Google Colab. The detector achieved a mAP accuracy of 95.6%. The dataset used in this research is the BIT-vehicles public dataset, that includes six classes of vehicles taken in different times during day and night. The model is then be implemented on PYNQZ2 FPGA. This development board from Xilinx support python coding through jupyter notebook.The board also has the capability to be connected to a real-time camera to feed the system with live input from different busy intersections. Depending on the result of the detector, the traffic light system can be controlled. When objects are detected, the light is turned red while turning the opposite side's light to green. This way a car does not have to wait at a red light when there is no object coming from the other direction. Implementing such smart system can reduce traffic jams at busy intersections. As a result, saving time and resources in cities.The work presented an AI-optimised solution that is cheap to implement on FPGAs and it does not involve extra hardware devices such as a computer or a radar system. This can also play a key role in improving mobility and reducing waiting time as well as the emissions caused by polluting gases.

Contents

Acknowledgments

I would like to thank my Supervisor, Dr. Olufemi Isiaq for providing guidance, helpful feedback and making me confident in my abilities throughout this project.

Acronyms

CNN	Convolutional Neural Networks
GPU	Graphical Processing Unit
mAP	mean Average Precision
IoU	Intersection over Union
TP	True Positive
FP	False Positive
FN	False Negative
YOLO	You Only Look Once
MMT	Million Metric Tons
FPGA	Field Programmable Gate Array
API	Application Programming Interface
PYNQ	Python productivity on Zynq
DVI	Digital Visual Interface
ASIC	Application Specific Integrated Circuit
AI	Artificial Intelligence

1 Introduction

Traffic jams caused by inefficient traffic signal control are a prevalent issue in big cities and busy intersections. While cars are stuck in traffic for hours, greenhouse gas emission increases air pollution. Also, it is estimated that this delay causes the United States economy to lose approximately 87 billion dollars annually (Weforum, 2018). Therefore, it is necessary to have a system that makes the traffic signals control smart, especially since most cities are advancing towards a smart city concept.

Transportation experts predict smart traffic signals are necessary for the operation of autonomous driving cars. Cities must adapt their infrastructure to withstand the increased number of vehicles on the road. The cost of fuel and time due to traffic congestion was estimated to be around 78 billion dollars in 2005, and 2.9 billion extra gas gallons were bought just because of traffic in the United States (Texas A&M Transportation Institute). A new study in Hampshire, United Kingdom, predicted that using smart traffic systems based on real-time data will save 205 Million Metric Tons (MMT) from global emissions by 2027, nearly double today's 145.7 MMT (Juniper Research, 2022). Many cities are increasingly adopting smart intersections. It is estimated that the investment into smart intersections will reach 10.2 billion by 2027 from 5.7 in 2022.

Researchers state that the air pollution we inhale from traffic leads to an average of 60 -70 asthma conditions (Cambridge University, 2005). Multiple solutions have been proposed over the past two decades to mitigate these risks. The smart signal system is the most common suggestion since it constantly monitors traffic to reach a smoother vehicle flow. Such system implements artificial intelligence, smart cameras, radars, and sensor decoders into traffic infrastructure. In Pennsylvania, researchers started their Surtrac traffic control pilot system using real-time sensors that get their information from a software integration/API. They then create an optimization plan and act accordingly by sending commands to the controller to coordinate the signals (Smith et al., 2013). Others, such as Tal Kreisler and Uriel Katz, developed technology in Tel Aviv called" Plug and Play", which uses Virtual Management Center (VMC) in the cloud combined with human eye-level sensors installed at each intersection around the city. The software costs 20,000 dollars per intersection. However, sensor data was largely limited to highways and primary roads because the sensors were typically installed only on the most heavily travelled or traffic-prone routes. Google Maps used traffic layers to combine data from cars using GPS to be sent back as a coloured line.

Convolutional Neural Networks (CNN) architecture differs from standard Neutral Networks (Frank Emmert-Streib et al., 2020). It replaces the use of neurons connection between layers; instead of continuous fully connected neurons in neural networks, CNN decreases the number of parameters within each layer by making convolutional layers that are not all connected to neurons in the next layer. CNN can have one or many of these convolutional layers (Frank Emmert Streib et al., 2020). It is a special type of neural network that is designed to process images and videos. The convolution process is basically choosing a filter with a known size and a task, for example, a 7*7 filter for edge detection and having this filter convolve over the image pixel by pixel. In terms of image processing, it has proven to be highly accurate as well as quick with analysis and image feature extraction. Width, height, and depth combine to create a matrix that can pass a CNN so the structure will always be preserved. (Y. LeCun and Y. Bengio 1995) The core building of CNNs is the convolutional layers (Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, 2015). All these layers are responsible for detecting the patterns inside the image when filters are applied, such as the kernel containing the image pixels. The dot product of these filters is the convolution which will enable the network to learn from low-level features such as lines and edges, then high-level features for the objects in deeper layers like faces. As a result, this mechanism will reduce the tasks of feature extraction development.

1.1 Objectives and Problem Statement

Multiple technologies, including thermal cameras, radars, lasers, and sensor detectors, have been used in traffic lights to detect vehicles and reduce congestion. However, the mentioned technologies are still struggling to adapt to the rapidly increasing number of cars on the road. Deep learning emerged as an effective and accurate tool for object detection, enabling it to be used in automating traffic light signals. The most famous deep learning algorithms require powerful hardware, such as GPUs, to be trained and run in real time. One possible solution to fill this gap is to deploy a lighter version of the deep learning detection model in an FPGA. That way, it does not need a computer to run. When the FPGA is connected to a smart camera, it can be mounted on traffic lights to make them ’smart’.

The smart city concept has been used very widely recently. The proposed system aims to make traffic lights smart based on detecting cars using the deep learning model. The model can be trained using a public annotated cars dataset to detect cars, people, and bikes. The Tiny YOLOv3 tiny is a smaller version of YOLOv3 that can be implemented in FPGA (AlexeyAB).

The FPGA can then be connected to a smart camera already deployed in most intersections. When the system detects a car, person, or bike, the signal should be changed to red while having the opposite road's signal turn green. The system could cut the time cars wait when the signal is red and no cars are passing from the opposite side.

The majority of the work done within this area involves using complex and expensive hardware such as radar and sensor networks in order to automate the signals. However, none of the systems took advantage of deep learning which can optimize this problem rather than the existing methods in terms of multi-class and has the highest detection accuracy such as the Tiny YOLOv4. In addition, this detection algorithm has the advantage of fast real-time processing, which is necessary for our problem. The proposed system can be deployed in busy intersections to automate the traffic signal based on the results of detecting objects, which makes moving cars' flow more efficient.

The work in this thesis investigates the following research question: *How can deep learning based object detection, implemented in FPGA, make traffic light control systems smart in order to reduce traffic congestion?*

2 Background

Smart Traffic Systems (STS) Recently, different technologies have been used for traffic lights system other than the traditional controller. The rapid increase in the number of vehicles has led to major problems in distinct areas, such as the negative environmental effect. The increased number of vehicles in traffic increased the carbon emission levels from the engines, directly polluting the environment (George Milev, Astley Hastings and Amin Al-Habaibeh, 2019).

When the English railway manager John Peake (J. P. Knight) invented the first manual traffic light system on railway signal lights in 1868, the main purpose was to reduce

the number of accidents on the road. Then, in the 1960s, smart traffic lights came into the picture when computer technology started modernising. The goal of developing traffic systems has always been to improve traffic flow by detecting and reducing congestion by adding intelligent sensors. These sensors and technologies are integrated into highways to manage traffic in real-time by collecting, processing, and analysing traffic information. Examples of these advanced traffic management systems are (Aryaomnitalk, 2020) and (Trafiksol, 2018). The advances in Artificial Intelligence (AI) and the use of deep learning made the use of real-time data possible in these systems. Advanced Traffic Management Systems (ATMS) use real-time data compared to historical data trends to create a relationship that supports predictions based on AI algorithms (Swarco). In order to achieve the concept of smart cities, the EU commission evaluation depends on four major areas: buildings, electricity, cooling/heating systems, and transportation (Vito Albino, Umberto Berardi and Rosa Maria Dangelico, 2015). Deep learning makes it possible to process a huge amount of real-time traffic data quickly and accurately (Mahashreveta Choudhary, 2019) described the smart traffic light system's importance in smart city initiatives. They highlighted the importance of intelligent transport system application and that it is not limited to congestion but can also be used to improve the efficiency and safety of the roads. The National Highway Transportation Authority (NHTSA) reports found that more than 40,000 people died in 2016 in the United States due to traffic crashes, with half a trillion dollars in annual expenditures from traffic congestion.
Another benefit of the traffic safety improvement and green applications include gas and energy minimisation (Rekor, 2022) raised 2.3 million dollars in funding in Silicon Valley to help cities optimise traffic management systems relying on instant access to predictive analytics. All the discussed analysis highlights the significance of the traffic congestion management problem and its impact on the future of our cities, especially after the (United Nations, 2018) reports about the dramatic increase of population in urban areas by 68 % in 2050.

2.1 Object detection

Object detection is a well-studied topic in computer vision. Traditional computer vision algorithms depend on human-defined shapes and angles that determine how accurately the features can be extracted. The emergence of deep learning and Convolutional

Neural Networks (CNNs) effectively changed the way of object detection to produce more meaningful results by learning complex features of objects with remarkable accuracy compared to traditional algorithms (LeCun et al., 1989). For example, if a human looks at a photo, its objects can be automatically recognised by the human brain in a matter of seconds. The same concept applies to object detection in deep learning when the neural networks replicate this kind of intelligence.
Object detection can be implemented in smart traffic systems by detecting cars, bikes, and pedestrians. The huge number of public datasets of vehicles makes it possible to train deep learning models to detect these classes. A fast and easy way to implement the real-time algorithm for object detection using deep learning is YOLOv4 Tiny. It is a simplified version of 7 YOLOv4 that includes only 15 convolution layers, making it possible to run on embedded sensors such as FPGA. The algorithm boasts high-performance levels, fast detection, and real-time tracking (AlexeyAB).

2.2 Deep learning in FPGA

Deep learning in Field Programmable Gate Arrays (FPGAs) are integrated circuits with a hardware fabric. It has become an important technology in the field of AI applications because of its ability to run CCNs with low latency implementation and much less power consumption. Unlike the circuit in GPUs, the FPGA chip can be reprogrammed as needed. Due to this feature, FPGAs have become an excellent alternative to application-specific integrated circuit ASICs which are one-time programmable and require more time for development (Ian Kuon, R. Tessier and Jonathan Rose 2008).
The technology industry has recently adopted FPGAs for deep learning. For instance, in 2010, a detailed report from Microsoft showed one of the first and most significant use cases of artificial intelligence with FPGAs. It has been an instrumental tool for accelerating web searches. FPGAs offer optimum speed, easy programmability, and delivery flexibility at a much lower cost than Application Specific Integrated Circuits (ASICs). After a while, the company relied on Microsoft's Bing search engine to use FPGAs in production, and this was the best evidence of the importance of their application in the fields and applications of deep learning.

Using FPGAs to speed up search rankings, Bing achieved a massive productivity boom at nearly 50FPGAs, which have also been used in object detection. Researchers presented an implementation for a compressed YOLOv3 tiny on FPGA SoC to reduce the memory size by 75 percent of 104.17 FPS. Their real-time detection was about three times faster than the embedded GPU performance (S. Oh, J. -H. You and Y. -K. Kim, 2020). Their findings in the study show that FPGA is suitable for deep learning applications. Other example of implementing deep learning-based algorithms on FPGA is is presenting by (Bao et al. 2020). They used the Winograd algorithm for YOLOv2 but with Zynq FPGA The researchers compared their test results with the implementation of YOLO in a GPU and in Zynq Ultra scale+. Their proposed method has maintained accuracy, saved resources, and reduced power consumption on accelerating deep learning networks. Moreover, (E. Raze, A. Khanaev and A. Amerikanov 2021) proposed a smart camera connected to DE10-Nano FPGA for detecting objects on a 128×128 video stream by a neural network in real-time using BlueOil framework. The final test resulting 28.3 to 33.4 FPS rate which is efficient but not the desired accuracy. (D. Goshorn et al. 2010) presents a high-performance FPGA implementation of a generalized parts-based object detection and classifier that runs with capability of 266 frames per second.

The limitations of the discussed research are that none implement traffic light optimisation in smart cities. The potential of deep learning-based object detection in FPGA can be further investigated in order to solve the problem of traffic light control systems. The system we propose in this thesis is to train YOLOv3 tiny real-time object detection model on a CPU laptop first. Then the trained model will be deployed in FPGA. The FPGA is then connected to a smart camera to capture real-time frames from highway intersections. The proposed system will be evaluated on a prototype of a traffic light. The framework architecture for deep learning is Darknet (darknet), in our VM, such as Google Colab. A virtual FPGA can replace the FPGA hardware. The FPGA vision remote lab is an open educational lab based in Germany that allows researchers to evaluate the design remotely and provide the results from a server working on the FPGA board (Marco Winzker, 2022).

3 Related works

The use of object detection for CNNs on FPGA has been an active research area (Tu et al., 2019). These technologies develop deep learning inferencing with FPGAs and GPUs combined with real-time multimedia applications because they do not want to design the Xilinx Artix-7 FPGA hardware implementation. The results from their framework achieved faster computation and lower power consumption. (Edward, Anton and Aleksandr 2021) discussed the integration of the YOLOv3 tiny object detection algorithm on DE10-Nano FPGA board, the aim was to solve the real-time video streaming object recognition problem. Results of training on BlueOil framework showed for different objects have a good percentage of mAP = 29.4% and FPS (around 30 frames per second with 128*128 input images) which is suitable for real-time applications. Limitations that they used OpenImagesV4 dataset which had 600 classes and a huge of tagged images (2 million), and they did not implement it on a traffic light system in their study. (S.Vamshi Krishna and P.Giri Prasad 2017) In their study, they proposed an Intelligent Transportation System (ITS) to identify traffic management on each road by using IR sensors to detect the traffic density implemented on FPGA. They managed the traffic flow using a VHDL simulator called Model Sim for testing the system. The system has been verified and simulated. (R. V. Kshirsagar and V. V. Dabahde 2015) proposed a Traffic light controller (TLC) system based on a microcontroller and microprocessor to minimise waiting times of vehicles at traffic signals. They designed a system that uses the traffic decoder sensors and the FPGA platform using VHDL. The system has numerous benefits over the exciting TLC and has been implemented effectively and evaluated in hardware (2015). (S.L.Qaddori and N.T.Gadawe 2020), proposed a combination of FPGA and Arduino Mega board for the traffic control systems. The system was programmed using VHDL for FPGA and C++ to programme the Arduino platform, which is used as a real-time simulator of traffic light systems. They concluded a comparison with the Arduino prototype model, which is economical, user-friendly, and works as a standalone. At the same time, the FPGA 12 Spartan 3E platform has high security, reliability, efficiency, and great speed. (D. Goshorn et al. 2010) present a high-performance FPGA implementation of generalised parts-based object detection and classifier that runs with a capability of 266 frames/sec. When it comes to deep learning, a dataset is an essential component in order to train and test the models. A set of papers works on various datasets, each of which has its own features. Careful research on the importance of finding a suitable dataset that serves our system and what the limitations will be for not using many of the discussed related works.

Researchers (S. Oh, J. -H. You and Y. -K. Kim 2020) used tiny yolov3 on FPGA for realtime detection, but the dataset they used for implementation is not for cars. Their dataset used pedestrian signal images (2020). The pedestrian signal dataset consists of the green signal images and the red signal images, and this is not what we are trying to detect. (S. Rujikietgumjorn and N. Watcharapinchai 2017), used UA-DETRAC dataset for RCNN architecture vehicle detection, which has similar categories to the COCO dataset. They use sub-classes output and pre-trained weights. These pre-trained weights take them directly from COCO dataset. Since the test dataset (UA-DETRAC) is like some categories in the COCO dataset, they used a COCO pre-trained model for fine-tuning the full network with UA-DETRAC dataset, which is too lengthy and does not have a pedestrian category.In their research (X. Li et al. 2019), proposed The ParallelEye Dataset, which has a large collection of virtual Images for traffic vision and consists of seven sub-datasets. The Pipeline for generating the ParallelEye dataset with OpenStreetMap, CityEngine, and Unity3D complicated it. In comparison,(M. P. Philipsen et al. 2015) are trained in traffic light detection, not for cars. (Z. Huo, Y. Xia and B. Zhang 2016), applied the EdgeBoxes algorithm and random forest classifier to generate candidate vehicle objects.However, the dataset is not annotated and needs to be reformatted with YOLOV3-TINY.

(Jianxiao et al. Dec 23) propose a multi-sensor multi-level composite fusion network for robust multi-scale vehicle detection under variable lighting conditions, but the dataset availability is on request. The dataset is not publicly available now due to deficient maintenance capacity. Gong et al. proposed an improved yolov3-tiny structure which boosts the network speed by adding a series of 3 × 3 and 1 × 1 convolutional layers to the original yolov3-tiny as well as mAP improvement by 6%. The experiment used a dataset called FLIR ADAS, but all the images are acquired with a thermal camera, and this is the reason we cannot use their dataset with our system.
In their study,(D. T. Nguyen et al. 2019) discussed the hardware implementation for realtime object detection using VC707 FPGA on PASCAL VOC 2007,2012 dataset and tested using PASCAL VOC 2007 in CNN YOLOv2 tiny. Their work achieved 64.16 mAP and was cost-efficient in synthesis design. (T. Fang, X. Huang and J. Saniie 2021)

proposed a face mask detection using PYNQ SoC platform implementied in Zynq-7020 SoC device. Haar cascade classifier was used as a detection algorithm in real-time environment.In 720P output achieved a good percentage of FPS running at 45.79 resulting 96.5% mAP. Their detection algorithm is too old and they did not involve any Deep Learning techniques.

Moreover, By using DL and OpenCV (a library of programming deep learning models aimed at realtime computer vision scenarios). (Chandan et al. 2018) present a system solution for real-time detection. SSD algorithm implementation with MobileNets in the python environment was set to detect objects in motion. The trained model is designed to detect 21 classes with an accuracy of 99 per cent and a considerable confidence level. In another study by (Z. Chen et al. 2019) they introduced a system for smart mobility applications such as traffic roads in two approaches. YOLO v3 and SSD have both been tested and compared in a traffic environment. According to their work, YOLOv3 is faster than SSD and performs better overall. Furthermore, the precision of the YOLO Algorithm is higher, especially in real-time.

3.1 Computer Vision and FPGAs

Recently, the FPGA family made an innovative technological advancement, combining all the necessary components for a CPU in one chip called System on Chip (SOC). This integration allowed the programmable logic to interact with the processing system by C programs. In addition, the High-Level Synthesis Tool (HLS) has been used by Xilinx in a platform called Vivado to accelerate C codes to a Hardware Description Language HDL. This means the open source computer vision library, OpenCV, can now be used in FPGAs. In their research, (Henrik 2015) provides a thorough description of how Vivado HLS works on OpenCV functions using the Zynq-7000 FPGA board. Furthermore, a recent study redesigned YOLOv3 tiny deep learning architecture to be deployed on face detection problems using the FINN framework and FINNHLS library. This includes the layer definitions in C++ in PYNQ-Z2, which is used as a target board for implementation. The study shows the accuracy difference between using a Quantised model such as FINN in various bit width precision with non-Quantized. The results show that the quantisation models led to 50 times faster detection with a tolerable accuracy drop than non-quantised ones (Bestami, Sefa and Hasan 2022). (M. Dhouibi, A. K. Ben Salem and S. B. Saoud 2020) used PYNQ on ZedBoard framework deployment for object recognition. By achieving 100ms latency and up to 10 image recognitions per second on the CIFAR-10 dataset with 79.90 per cent accuracy, their study compared the performance to the software implementation on both CPU and GPU. Their findings proved that PYNQ is an effective solution for deep learning a wide range of embedded applications while remaining efficient in energy consumption. Also, (Longzhen Yu et al. 2022) have used PYNQZ2 FPGA board in their study to propose an efficient algorithm of YOLOv3 Defect Inspection based on an attention mechanism. Their experiment results showed that the algorithm can be deployed with high efficiency and good accuracy reaching 99.2%, processing speed reached 1.54 Frames per Second (FPS), as well as very low power consumption.

4 Theory

After having the pilot experiments dealing with the YOLOv3 Tiny object detection algorithm, the model desired accuracy was not achieved after many tries. However, the YOLOv4 Tiny version had a faster processing speed with better accuracy (around 7 per cent) and is 10£ more than the older version with 12£ FPS. The architecture of YOLOv4 has four blocks: (Backbone, Neck, Dense Prediction and Sparse Prediction). Following these distinct blocks, the backbone for this version is the Cross-Spatial - Partial CSPDarknet53, which has the ability to split the current layer into two parts, one will pass through the convolutional layers, and the other won't. Then the results will be aggregated before they pass to the Neck block. The Neck is responsible for adding layers between the backbone and Dense Prediction block by using the Path aggregation technique and pyramid pooling to improve the accuracy. The Dense Prediction block is considered the head of this algorithm. The primary role of this block is to locate the bounding boxes for the classification process. The bounding boxes are (x, y, height, width) coordinates. This will divide the input into grids, and the prediction of each grid cell contains an object by using anchor boxes to provide an output with probabilities of the class vector with the bounding box. Many other techniques are used internally in YOLOv4 algorithm, such as bag of specials and bag of freebies.

These are responsible for improving the accuracy during and after the training of the model. The interface time is not increased during the training process when the algorithm uses bag of freebies. However, the bag of specials minimally increases the interface time and changes the architecture of the model by using two special techniques. The first is mish activation and cross-stage partial connections. The second is the bag of specials for detection, which uses the SPP block, the SAM block, and others. According to research conducted by Byung-Gil Han et al., results for training the newer version are not the same. From the first try, the YOLOv4 tiny algorithm achieves a greater accuracy percentage with the same interface time running on a GPU.

4.1 Image classification

Many image classifications predict objects in images by training a multi-label classifier which will be responsible for predicting classes for any object inside these images. After the classification process (if the class is known), the location of an object will be determined using the Localisation method. If not, the (object class is not known), then the model should predict not only the location but also the class for each object. This process is called object detection. Multi-class object detection problems occur when multiple objects are inside a single image. The most used state-of-the-art methodologies to train object detectors are these methods which use CNNs and deep learning.

4.1.1 Region-based Convolutional Neural Networks(R-CNN)

Convolutional neural network-based classifiers were first used in the RCNN algorithm instead of HOG-based classifiers, where the success calculated the accuracy of the classification. RCNN used an algorithm called selective search to solve the problem of object detection by reducing the number of bounding boxes that fed the classifier to region proposals to generate all the possible locations for an object inside an image. And then, fed these boxes to the CNN classifier with fixed size 224*224 followed by SVM to predict the class for each patch and optimise these patches by training bounding boxes separately.

4.1.2 You Only Look Once

For YOLO, detection is a simple regression problem which takes an input image and learns the class probabilities and bounding box coordinates. This makes it become a prevalent choice for object detection problems. YOLO divides each image into a grid of S x S, and each grid predicts N bounding boxes and confidence. The confidence reflects the accuracy of the bounding box and whether the bounding box contains an object (regardless of class). YOLO also predicts the classification score for each box for every class in training. Both classes can be combined to calculate the probability of each class being present in a predicted box. So, total S x S x N boxes are predicted. However, most of these boxes have low confidence scores and if we set a threshold, say 30% confidence. This unique structure makes this algorithm much faster than the aforementioned network structures. To improve accuracy and speed of detection, YOLO author proposed YOLOv2, which replaces the fully connected layer with a conventional layer. In 2018 for further improvements, YOLOv3 was released. In the YOLOv3 model, 35 convolutional layers are used for feature extraction inside the DarkNet-53 backbone. Then, YOLOv3 was used as the head network for YOLOv4. YOLOv4 replace the previous backbone with some extra features by using a modified version called CSPdarknet-53, where CSP cross-stage-partial connections work on the feature extraction separation into two parts. Inside this version, the Mish activation function is used (Diganta Misra 2019).

Recently, YOLOv4 tiny has proposed to improve real-time performance. This network model is a lightweight algorithm for YOLOv4. The main difference is this model is designed especially to train on machines with less computing power. We can see that the model's weight is around 16 megabytes. The interface speed on YOLOv4 tiny is 3 ms tested on Tesla P100, making it one of the fastest object detection algorithms.
CSPdarknet-53 backbone uses this version's three residual modules from 29 compressed convolutional layers, and the leaky ReLU is used as the activation function. This network uses two detection heads for classification and regression of the prediction, which means YOLOv4 has relatively competitive results compared with YOLOv4 due to the size reduction. The COCO dataset achieves 40 mAP with fewer anchor boxes for prediction. All the above facts clarify the power of this detection system and why it

sets the high water mark for object detection. YOLOv4 tiny is considered a better option, especially in a real-time object detection environment, as fast interface time is more important in our experiment than accuracy or precision.

DarkNet Model conversion

After we trained the model using the Darknet framework and obtained the required accuracy and speed, we had to convert the workflow to a Tensorflow environment. The next step requires executing on the FPGA board and running the model using Python. There are multiple ways to convert the model to a Tensorflow environment so that it can be implemented on the PYNQ board and compatible with Python. There are some helpful scripts for this in the Keras-YOLOv3 repository (David, 2022). Using Conda, the code has been successfully converted to the new framework and tested on more than one image. The default version of Python of the PYNQ latest image is 3.8. We did the model conversion beforehand.

4.2 PYNQ Development Board

PYNQ board is an open-source software framework designed around Xilinx Zynq SoC with A9 dual-core processor and FPGA. The board is designed to work on any computing platform and operating system. First, we tried to use the virtual lab for our hardware implementation. However, the limitation when running our experiment is that a YOLOv4- tiny DarkNet trained model was difficult to convert to a design language that every FPGA use, such as VHDL.

There are many alternative solutions to use in order to avoid the complexity of using other programming languages. One of these ways is PYNQ software. PYNQ is the first system that combines a high-level productivity language with FPGA overlays. The key advantage is that the overlay supports Python APIs and offers an alternative compiling prototype environment for designing and programming embedded processes without using other logic circuit design tools.

On the other hand, this version of Xilinx offers adopted a web-based architecture, which means PYNQ can incorporate an open-source infrastructure called Jupyter notebook framework to run Python kernel directly on the ARM processor. Also, the web server can access the kernel with many tools such as bash terminal and code editor.

This python-based approach for hardware control will be extremely useful to implement on our model by changing our used environment from DarkNet to Tensorflow, which has a broader scope. Thanks to Trinh Hong Trieu, all DarkNet models can now be converted to Tensorflow custom neural network framework and installed on the Windows environment. The development board below TUL PYNQ – Z2 Basic Kit, Zynq SoC has been ordered from Zynq official Spenser in Europe, Farnell UK company. PYNQ-Z2 FPGA-based platform belongs to the ZYNQ XC7Z020 FPGA family (Xilinx, 2022).

The board provides an HDMI In and Out, a mini display port out and 4 USB downstream ports. These features are essential for applications like ours to interface with images or cameras that can be connected to them. More information about the PYNQ project can be found at (Louise Crockett et al. 2019). PYNQ uses Jupyter notebook portal as an interactive computing environment, enabling users to author notebook documents that include the code, Images and Videos. 9090 is a port used by Jupyter servers to connect a terminal. With Jupyter web application, we can easily edit and run the code in the browser, which PYNQ associates with IPython kernel that runs Python code. (PYNQ 2018).

4.2.1 Pynq Image

PYNQ image is a pre-compiled bootable Linux image that includes the PYNQ Python required libraries, packages, and other open-source packages. To port the PYNQ image on a Zynq Device such as PYNQ-Z2, an SD card needs to be modified and adjusted to run Linux (Ubuntu 18.04) on the ARM core processor v2.7.0. After installation, this image will create the Zynq BOOT.bin, the u-boot bootloader, the Linux device tree blob, and the Linux kernel.

The following elements are a summary for the main reasons behind choosing PYNQ board:

- The use of Python as high-level productivity language.
- Python libraries exposed as FPGA overlays with extensive APIs.
- Embedded processors to serve web-based architecture, and
- The use of Jupyter Notebook framework deployed in an embedded context.

4.2.2 Overlays

An overlay is a term for hardware system that is programmed into and forms part of the processing logic PL. The hardware layer of the PYNQ framework. we can use existing overlays which are available on PYNQ project community directly to the board or by modifying or developing our own custom overlays. The most powerful part of overlays as opposed to regular bitstreams is their ability to interact with hardware designs from an existing Python code that will be running in a Jupyter notebook on a processing system. The PYNQ project creates a sample overlay. We used the Base overlay that contained logic to communicate with all external peripheral interfaces on the board. such as USB camera as an input device and HDMI out for display output. These connected interfaces may include switches, buttons,LEDs or audio connectors, Also there is a headers for for the Raspberry Pi, and Arduino within the board. Moreover, an additional component called a trace buffer which help the users send and receive signals by passing it through an external pins to be analysed and debug (PYNQ 2018).

4.2.3 Jupyter Notebook Interactivity

Jupyter Notebook is an open-source web application that lets us create codes containing inside the board using IPython project which can include visualisations. One of the most interesting aspects of Jupyter notebooks is being interactive. That means we can edit our code inside a web browser and integrate real-time viewing into their notebooks, Also, there are buttons to act as interactive widgets. viewing a web page from an external websites is also possible inside this environment. This makes it easy to modify any aspect of the code, redo it and provide results. This feature is also supported by the PYNQZ2 board even more interactivity is possible, this is because the Jupyter notebook contains interactivity aspects with the internal design components on its processing logic. This also include hardware implementation, that helps to run YOLOv4 Tiny algorithm.

4.2.4 Why OpenCV

To differentiate between Python and the other programming languages and why we chose this method in our implementation. First, the OpenCv library is one of the best open-source libraries which perform computer vision tasks. It is based on C++ functions and has Python bindings, providing faster execution time compared with MATLAB. For example, the implementation companies Java and C++. Moreover, there is a wide range of resources provided for image processing. The library has 'more than 2500 optimised open-source algorithms without licensing requirements. OpenCV is used in this project as it allows us to do the next step for the hardware implementation in the FPGA platform. (OpenCV, 2022), (Alberto Fernandez Villan, 2019), (Beyeler, 2017).
Several studies make use of this library in deep learning algorithms. (G. Chandan et al. 2018) used the SSD object detection algorithm to develop a python program implemented in OpenCV for the detection and tracking of a real-time video sequence. The model showed good results for 21 trained classes with an accuracy of 99 per cent. They concluded that it could be deployed in CCTV and surveillance systems. Another architecture used for the classification problem is MobileNetV2, which has been used with OpenCV Deep Neural Network (DNN) module and Tensorflow to classify faces for those who wear masks in real-time public areas (G. Harriat Christa et al., 2021) and (Nagrath et al., 2021).

4.3 Detection Evaluation

The evaluation metric used for the deep learning model is mean Average Precision (mAP) (Shivy Yohanandan). Specifically, the Pascal VOC 2010–2012 (Mark Everingham 2012), which samples a curve at all precision and recall values. The mAP is then calculated as the exact area under the precision-recall curve (AUC).
To determine whether a prediction is correct or not, Intersection over Union (IoU) is used. It is defined as the ratio of the overlap between the predicted and ground truth bounding box to the union of the two boxes. The objectness (confidence) score is the network's confidence that an object exists in the given box. A prediction is True Positive (TP) if: its objectness score is greater than or equal to some confidence threshold, the predicted class matches the class of the ground truth, and the IoU with

ground truth is greater than or equal to the defined IoU threshold. A prediction is False Positive (FP) if either of the latter two conditions is not true.

Precision is the percentage of TF among all predictions, and recall is the percentage of TF among the ground truths. The mAP metric in the Pascal VOC 2010–2012 interpolates all the points to calculate the AUC of the precision-recall curve. By default, the IoU threshold in this calculation is 0.5. Mathematically,

$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$

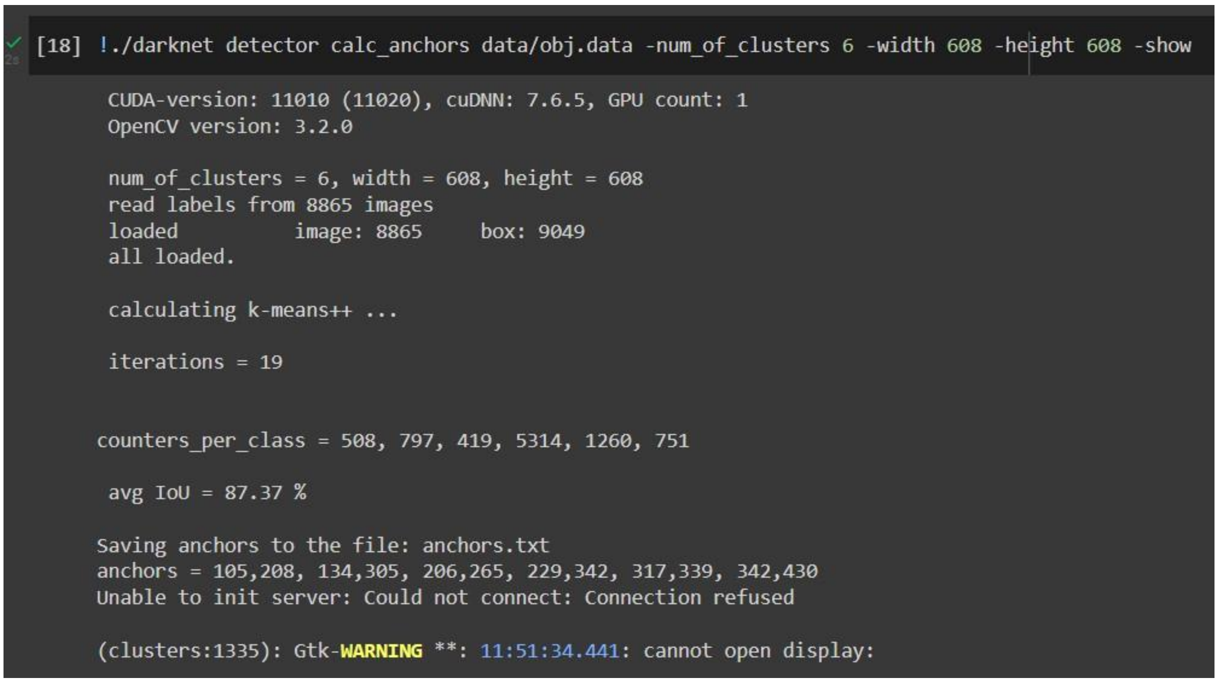
where FP and FN are false positives and false negatives, respectively.

5 Methodology

Our proposed system is to control traffic lights using deep learning-based object detection. A public annotated dataset is used to train a deep learning model to detect different classes of vehicles. The neural network we used is YOLO-tiny because of its ability to be implemented in FPGAs. The model can be trained using a virtual GPU or normal CPU.

The first step after the pilot study that was conducted earlier was upgrading the model with the same algorithm. YOLOv4-tiny is the updated version of our used model and has considerable improvements related to speed in real-time detection, promising accuracy and minimum loss. Also, FPS during real-time processing with YOLOv4-tiny on different hardware. YOLOv4-tiny ran much faster than the YOLOv4 model. We can see that the algorithm's processing with non-optimised weights was too slow for a real-world implementation. (Naeem Ayoub 2021) (Pavel Laptev et al. 2022) have more in detail comparison between Deep learning and YOLO types results in real-time object detection. FPS of YOLOv3-tiny and YOLOv4-tiny models on different image scales during real-time detection with optimised weights.

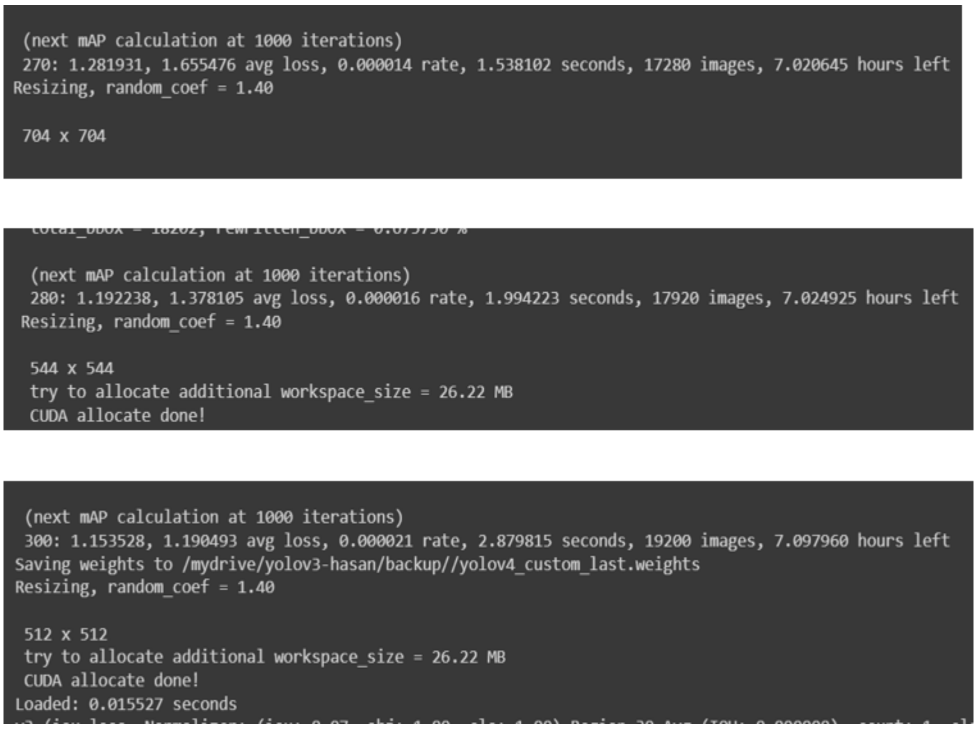
We train the YOLOv4-tiny on Google Colab, which is a free notebook that runs on a cloud server. It does not require any setup and the file is very easy to edit, just like google docs. The most important thing is that Colab supports many deep learning libraries. This makes it easier to import the data set, train the image classier, and evaluate the results (GoogleColab).



```
[18] !./darknet detector calc_anchors data/obj.data -num_of_clusters 6 -width 608 -height 608 -show

      CUDA-version: 11010 (11020), cuDNN: 7.6.5, GPU count: 1
      OpenCV version: 3.2.0

      num_of_clusters = 6, width = 608, height = 608
      read labels from 8865 images
      loaded          image: 8865     box: 9049
      all loaded.

      calculating k-means++ ...

      iterations = 19


     counters_per_class = 508, 797, 419, 5314, 1260, 751

      avg IoU = 87.37 %

     Saving anchors to the file: anchors.txt
     anchors = 105,208, 134,305, 206,265, 229,342, 317,339, 342,430
     Unable to init server: Could not connect: Connection refused

     (clusters:1335): Gtk-WARNING **: 11:51:34.441: cannot open display:
```

Figure 5.0.1: Anchors for used for the YOLOv4-tiny configuration file

The training set we used for the YOLOv4-tiny model consisted of 8865 images and 985 for validation. The dataset used is described in section 5.2. The network configuration includes number of hyper parameters that we adjust throughout multiple training trials to optimize their values. These include the following parameters: batch size, subdivision, network resolution, and anchors. These parameters are set as: batch size: 64, sub-division: 16, height and width: 608, and max batch: 12000. As the dataset has six class, the number of filters before each of the three YOLO layers is set to 33. The anchors were defined based of the sizes of the objects in the dataset. In figure 5.0.2 are the six defined anchors.
The training process took around 10 hours. The YOLO network does a random resize at different iteration during training to ensure the model learns from different image sizes. The figure below shows random re sizes of images to ensure the increase of model robustness.
After multiple training, the desired detection accuracy measured by mAP (section 4.3) was achieved. Using the validation set, the best mAP value was 95.6%. The next step in our experiment is to set up the PYNQ board and install the required dependencies in order to run our model in Python.



```
 (next mAP calculation at 1000 iterations)
 270: 1.281931, 1.655476 avg loss, 0.000014 rate, 1.538102 seconds, 17280 images, 7.020645 hours left
Resizing, random_coef = 1.40

 704 x 704
```

```
 (next mAP calculation at 1000 iterations)
 280: 1.192238, 1.378105 avg loss, 0.000016 rate, 1.994223 seconds, 17920 images, 7.024925 hours left
Resizing, random_coef = 1.40

 544 x 544
 try to allocate additional workspace_size = 26.22 MB
 CUDA allocate done!
```

```
 (next mAP calculation at 1000 iterations)
 300: 1.153528, 1.190493 avg loss, 0.000021 rate, 2.879815 seconds, 19200 images, 7.097960 hours left
Saving weights to /mydrive/yolov3-hasan/backup//yolov4_custom_last.weights
Resizing, random_coef = 1.40

 512 x 512
 try to allocate additional workspace_size = 26.22 MB
 CUDA allocate done!
Loaded: 0.015527 seconds
```

Figure 5.0.2: Random input images resize during training

5.1 SOFTWARE IMPLEMENTATION

The process of the software implementation was quite challenging. The training phase which resulted both the .weights and .confg files for YOLOv4 tiny model which are supported by C language as we use Darknet as a framework. The FPGA implementation step was supposed to be done virtually using an open-source education lab based in Germany. Due to time limitations and the complexity of building the entire neural network using VHDL, this option was not possible. PYNQ-Z2, an alternative FPGA board that supports python was the next option, figure 5.1.1.

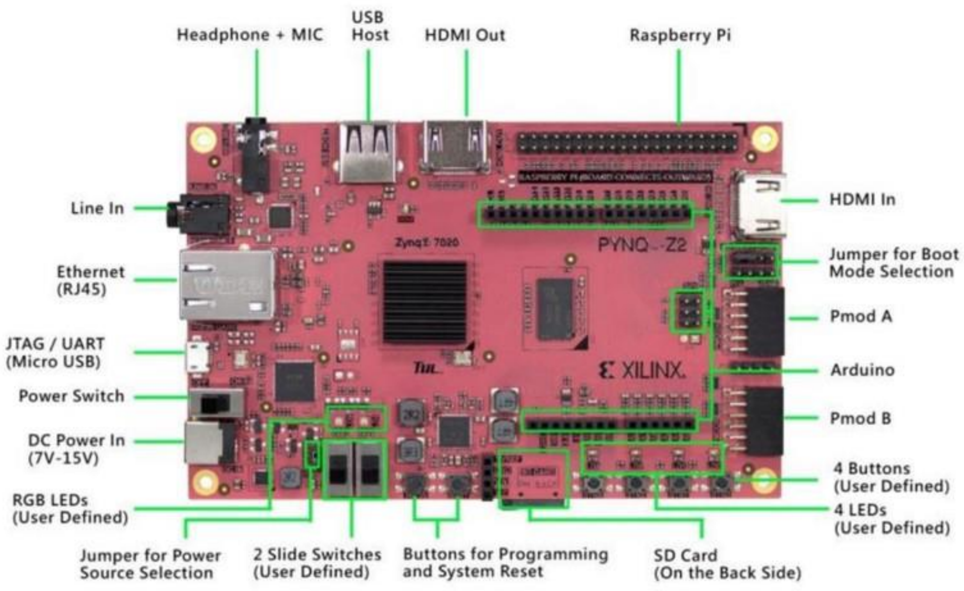


Figure 5.1.1: PYNQ-Z2 development board

5.1.1 PYNQ-Z2 board

The PYNQ-Z2 board is integrated with the dual-core ARM A9 processor. This makes the system design we aim to implement to be more efficient. PYNQ platform uses overlays which are included by default in the PYNQ image. The base overlay (Python productivity for PYNQ) is the one which supports the Python language, especially the OpenCV library, and includes multiple programmable logic circuits that can be called by the ARM core to be used as APIs (Pynq, 2018).

This PYNQ system functionality is very useful for users with no FPGA experience to interact with the board and all its I/O as it contains many features such as the video pipeline. PYNQ video pipeline is a module that uses IP blocks real-time that can pass an incoming video stream and generate an output video stream by interfacing with the HDMI input and output ports (Andrew Elbert Wilson 2020).

5.1.2 Board and System Requirements Setup

Following the official guide from PYNQ, first all jumpers should be set up and the board should be connected the LAN. The following steps were followed in order to set up the board:

- We used the official pynq image to be installed on the micro-SD card, running on latest version 2.7. It comes with the PYNQ Python supported libraries package.

- Tera Term desktop application was used as a terminal for pynq servers to configure the board.

SOFTWARE IMPLEMENTATION:

- The required libraries to be installed to the board are OpenCV-Contrib, Numpy and Matplotlib. However, the OpenCV-Contrib was not supported by the current version of python 3.8, so we had to downgrade the python version to 3.7 by building it from the source with the help of virtual environment.

- PYNQ overlays was very helpful to act as interfaces especially Video pipeline and the

  Base overlay. Also buffer and frame configurations were done for the HDMI ports and USB camera.

After setting up the board, the detector can be run in python. The Deep Neural Networks (dnn module) from OpenCV supports has YOLO-tiny implementation. The coding done to run the detector is described in detail in the artifact folder attached with the thesis.

To sum up our experimental set up, we trained the CNN model with the BITVehicle dataset in the Darknet framework. After the training, the .weight and .cfg are used for detection in PYNQ framework to perform using testing frames and an input test video. This step is done on a remote Jupyter notebook connected to the PYNQ FPGA web server. The inference was performed on a dual hardcore of the PYNQ board booted from a 32 GB SD card.

5.1.3 Implementation Challenges

Various challenges and limitations occurred regarding how to amend the code and find alternative functions that work with the Jupyter notebook. Moreover, installing the required libraries that the framework supports was not an easy task. PYNQz2 Xilinx project version runs with 32bit-bit dual-core ARM Cortex-A9 on it.So, one of the limitations faced was to downgrade the existing Python version of the PYNQ image from 3.8 to 3.7 to install OpenCV-Contrib 4.5.5 specialised module, which is an additional library that supports DNN functions. This stage took many hours to compile the library from the source and add it to the board. We also need to set them to point to the virtual environment libraries of the same type. moreover, HDMI Interface The PYNQ does not support the full HDMI protocol but supports the DVI protocol over the HDMI pins. The HDMI interface within the static logic uses IP blocks developed by Digilent and Xilinx to convert the DVI protocol to the VGA protocol and then finally to the AXI4- Stream protocol. Because the HDMI out clock was sourced from the HDMI in the clock and it may vary in speed more often, the HDMI output requires a working HDMI input which is not used in our experiment (Andrew Elbert Wilson 2020).

5.2 Datasets

An important aspect of the deep learning training process is choosing the dataset. Datasets affect the model accuracy and enormously. The bigger the dataset is, the more samples the network have to learn from and hence a more accurate model. We used two datasets to train two different YOLO-tiny models. The first is *dataset 1*, which was used during the pilot study to train the YOLOv3-tiny. As that dataset gave very low accuracy, we investigated available public annotated datasets. Therefore, for the YOLOv4-tiny model, we used *dataset* 2. A description of the two datasets is included below.

5.2.1 Dataset 1

This dataset is public and is published on a GitHub repo (Maryam Boneh, 2022) with YOLO format annotation is necessary for our training and evaluation. It included different angle views such as front and back with day/night views. This dataset consists of a total of 1376 images in five different classes: Car, Motorcycle, Truck, Bus, and Bicycle. It also contains the annotation files for each image in YOLO format.

5.2.2 Dataset 2

The Beijing Institute of Technology Vehicle Dataset (BIT-Vehicle) is a public annotated dataset that is available for traffic surveillance purposes and can work in complex scenes (Dong et al., 2015). We used this dataset to train the YOLOv4-tiny model. This dataset consists of 9580 vehicle images. There are six classes of vehicles which are: Sedan, SUV, Microbus, Truck, Bus, and Minivan. The number of objects in each class is 5922, 1392, 883, 822, 558, and 476, accordingly. The dataset includes daytime and nighttime images gathered from traffic monitoring cameras. The images were taken in different weather conditions, which makes them more robust.

Studies in the literature have used this inclusive dataset. In (Z. Dong et al. 2015), the authors used a CNN for vehicle type classification using the BIT-Vehicle dataset. Their network achieved an accuracy of 88.11%. The vehicle type classification method used a semi-supervised architecture model consisting of two stages, each including convolution, non-linearity, absolute rectification, normalisation, and pooling. These fully connected stages and the output predict the six vehicle classes.

Also, (Max N. Roecker et al. 2018) proposed an automatic vehicle type classification method using CNN on the same dataset. This classification model has 93.90% detection accuracy.

Furthermore, the work in (Jun Sang et al. 2018) proposed an improved model called YOLOv2Vehicle by improving the YOLOv2 algorithm to obtain better anchor boxes using the K-means clustering. Their experiment accuracy result is 94.78% using the same dataset. The study proved their method is effective for vehicle detection and has good feature extraction ability.

(M. A. Hedeya, A. H. Eid and R. F. Abdel-Kader 2020) Developed an ensemble of three deep neural network models for vehicle type classification. The networks are ResNet50, Xception, and DenseNet. Ensemble learning aims to supervise multiple models' strengths and weaknesses, leading to better classification. They used the BIT-Vehicle dataset to verify their results which were 97.6%. All the discussed work above used the same surveillance dataset for Vehicle Detection and Classification. However, none developed the model to be implemented in FPGAs.

6 Discussion and Results

The work proposed in this thesis aims to perform object detection based on deep learning to automate the traffic control process in smart cities. In the pilot study part of our work, a YOLOv3-tiny-based model was trained using dataset 1. That dataset was not big enough to train a deep learning model. The training dataset consisted of 1196 images to train, 125 for validation with five different classes as follows: Car, Motorcycle, Truck, Bus, and Bicycle. Therefore, the result of that model was bad. The detection accuracy measured by mAP for the validation set using the best training weights was only 32%, after 10000 iterations. Figure 6.0.1 below shows the mAP curve in red and the iteration curve in blue. Figure 6.0.2 illustrates the accuracy per class, along with the True Positive (TP) and False Positive (FP). We can see the accuracy is very low for all the classes, meaning the model missed detecting the objects.

As the main purpose of our pilot study was to investigate the research and overcome the problems we faced at that phase, that model needed to be improved. Therefore, we trained a new model using a new dataset, dataset 2. The YOLOv4-tiny has been proven to have higher accuracy than the YOLOv3-tiny . Moreover, we needed a bigger dataset as we have five different classes of vehicles to detect. Deep learning requires a significant amount of data to achieve high detection accuracy. Not to mention the angle of the frames and day/night vision.

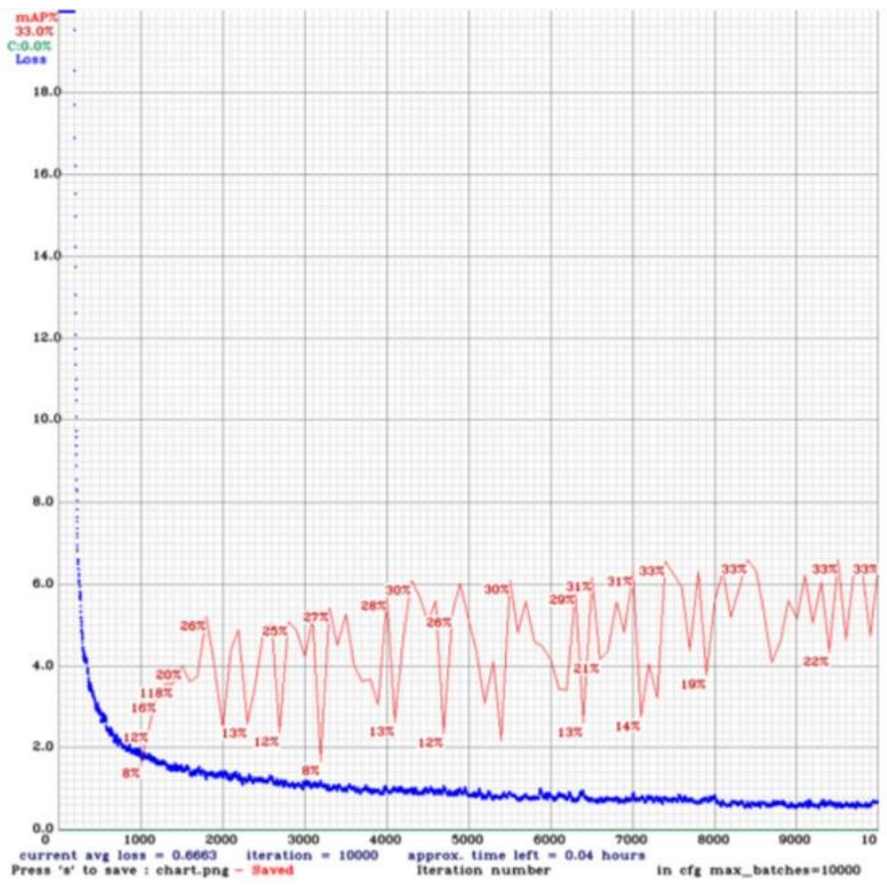
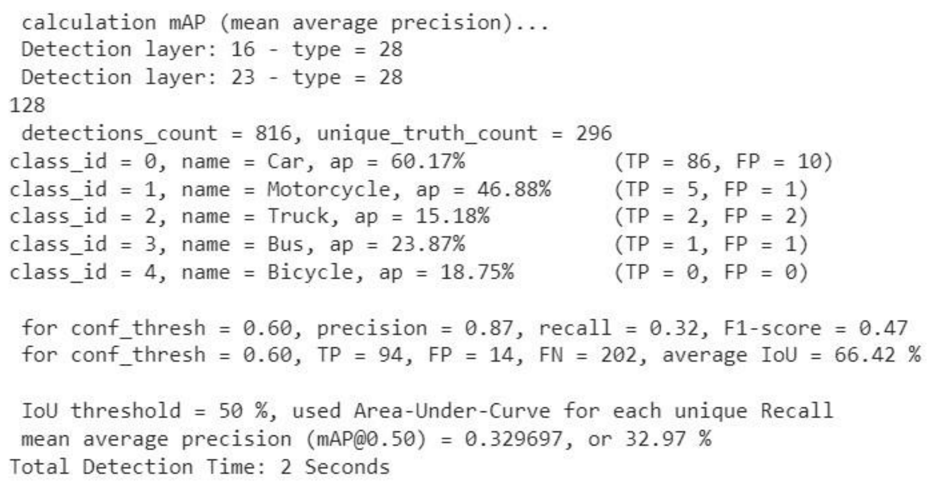


Figure 6.0.1: mAP accuracy of the YOLOv3-tiny model, trained on dataset 1



```
 calculation mAP (mean average precision)...
 Detection layer: 16 - type = 28
 Detection layer: 23 - type = 28
128
 detections_count = 816, unique_truth_count = 296
class_id = 0, name = Car, ap = 60.17%          (TP = 86, FP = 10)
class_id = 1, name = Motorcycle, ap = 46.88%   (TP = 5, FP = 1)
class_id = 2, name = Truck, ap = 15.18%        (TP = 2, FP = 2)
class_id = 3, name = Bus, ap = 23.87%          (TP = 1, FP = 1)
class_id = 4, name = Bicycle, ap = 18.75%      (TP = 0, FP = 0)

 for conf_thresh = 0.60, precision = 0.87, recall = 0.32, F1-score = 0.47
 for conf_thresh = 0.60, TP = 94, FP = 14, FN = 202, average IoU = 66.42 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.329697, or 32.97 %
Total Detection Time: 2 Seconds
```

Figure 6.0.2: YOLOv3-tiny mAP accuracy per class using dataset 1

Dataset 1 was the only available public option, but it did not give the desired outcome in terms of accuracy. The result from the pilot study concluded that the real time object detection requires more investigation in terms of choosing a compatible model with the FPGA and a good dataset.

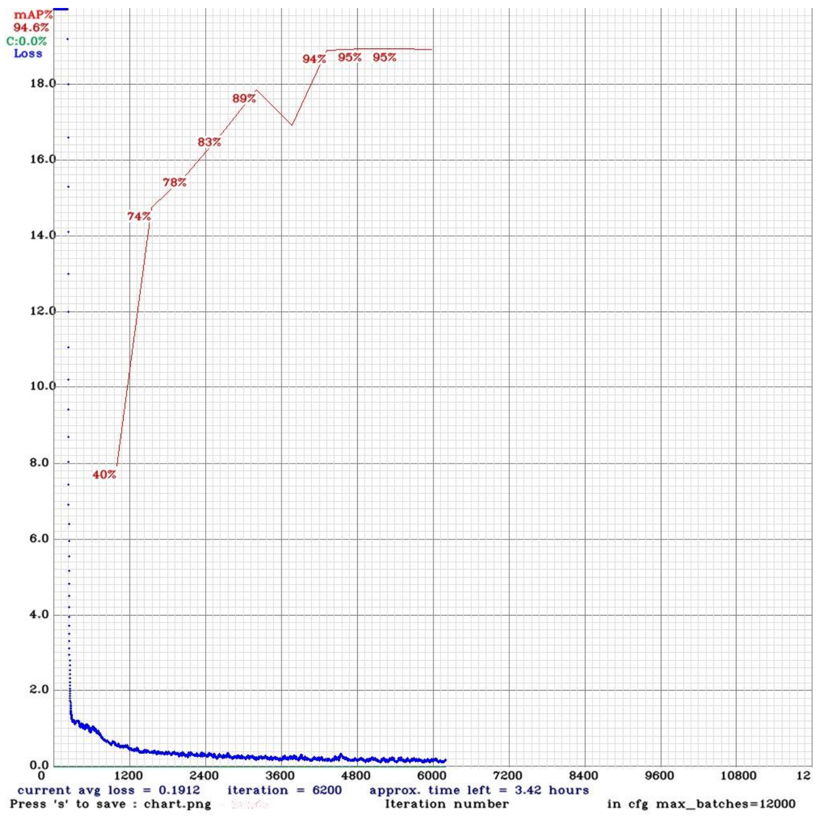
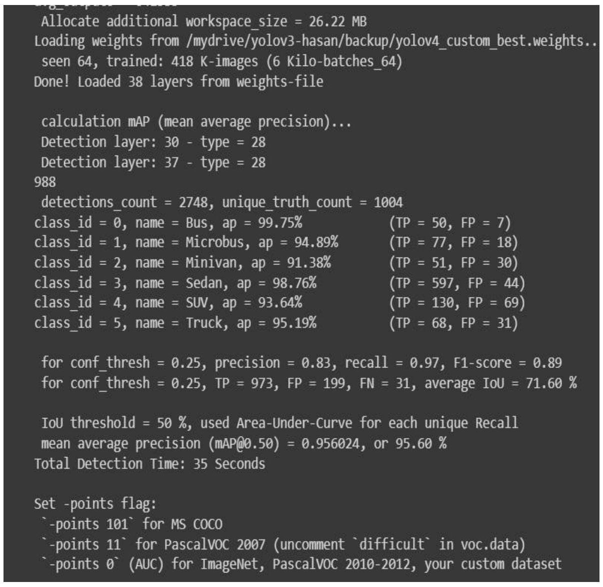


Figure 6.0.3: mAP accuracy of the YOLOv4-tiny model, trained on dataset 2

Therefore, a new model of YOLOv4-tiny was trained using 9000 objects divided into six classes. The model was tested using a validation set of 988 objects. The resulted accuracy measured by mAP was 95.6%, which is a big improvement. Figure 6.0.3 shows the mAP over 6700 iteration. Also, figure 6.0.4 shows the accuracy per class for the validation dataset.
We tested the model on both, image frames from the validation set and a short video. In figure 6.0.5 is an example of detection of random images from the validation set. All the vehicles were correctly detected including the night view frames. After Installing the required libraries detailed in the artifact (readme.txt), such as python, opecv-contib, DNN, and matplotlib into the PYNQ-z2 board, the darknet model was then used for detection through Jupyter notebook. The detector was run from the board and tested using testing images, such as in figure 6.0.5. Also, our python implementation in the board gives the option to run the detector using video feed and stream from a connected camera through

HDMI. However, when the frame rate was low when the camera was connected (6 FPS). This is due the limitation of the used RAM, which is 32.



```
 Allocate additional workspace_size = 26.22 MB
Loading weights from /mydrive/yolov3-hasan/backup/yolov4_custom_best.weights..
 seen 64, trained: 418 K-images (6 Kilo-batches_64)
Done! Loaded 38 layers from weights-file

 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
988
 detections_count = 2748, unique_truth_count = 1004
class_id = 0, name = Bus, ap = 99.75%           (TP = 50, FP = 7)
class_id = 1, name = Microbus, ap = 94.89%      (TP = 77, FP = 18)
class_id = 2, name = Minivan, ap = 91.38%       (TP = 51, FP = 30)
class_id = 3, name = Sedan, ap = 98.76%         (TP = 597, FP = 44)
class_id = 4, name = SUV, ap = 93.64%           (TP = 130, FP = 69)
class_id = 5, name = Truck, ap = 95.19%         (TP = 68, FP = 31)

 for conf_thresh = 0.25, precision = 0.83, recall = 0.97, F1-score = 0.89
 for conf_thresh = 0.25, TP = 973, FP = 199, FN = 31, average IoU = 71.60 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.956024, or 95.60 %
Total Detection Time: 35 Seconds

Set -points flag:
 `-points 101` for MS COCO
 `-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
 `-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset
```

Figure 6.0.4: YOLOv4-tiny mAP accuracy per class using dataset 2

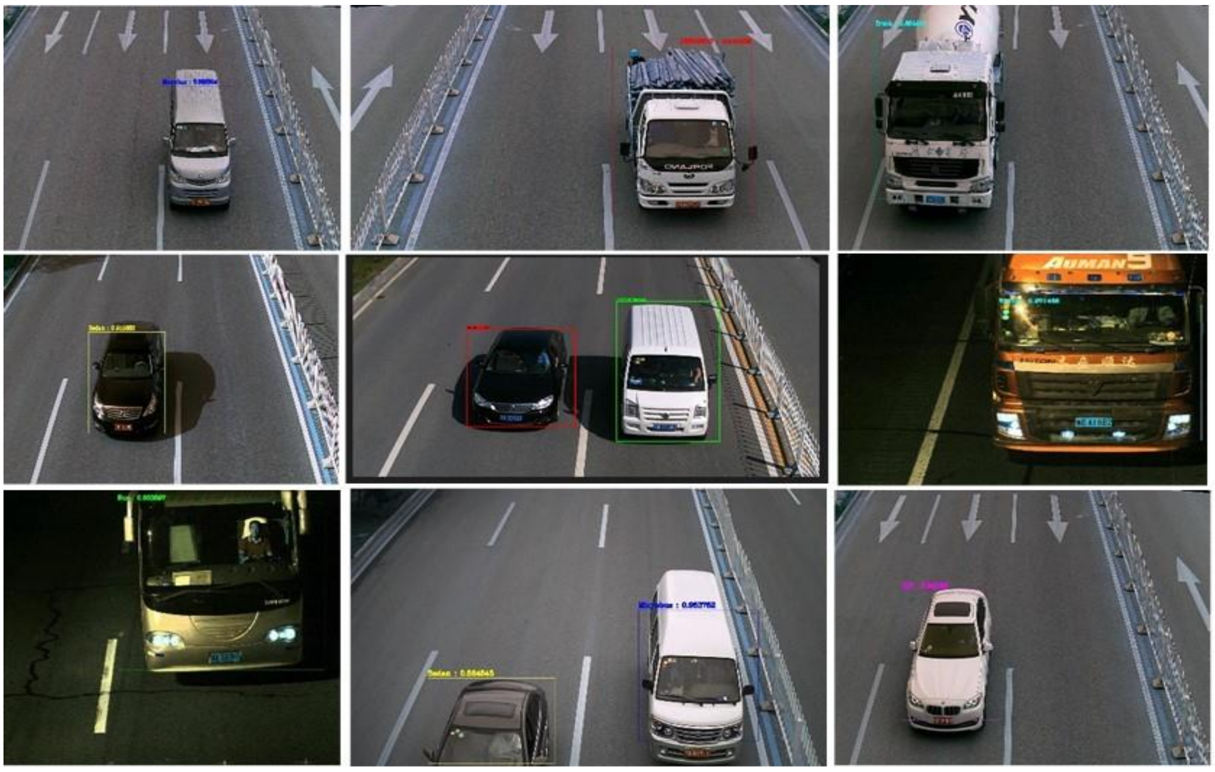


Figure 6.0.5: Detection Examples of YOLOv4-tiny model

7 Conclusion

Many cities around the world are in the process of making themselves smart. This means integrating the use of data and advanced technology to improve different aspects such as transport, energy use, and air quality to drive economic growth. The investigation done in this thesis is to answer the main research question of whether it is possible to implement deep learning algorithms into an FPGA to improve traffic management systems, with a focus on the concept of smart cities. Our work was conducted in two stages, a pilot study and a research project. The pilot study concluded that the deep learning-based object detection model (YOLOv3-tiny) we trained has low accuracy. The reason for that is the dataset was very limited, and the network did not learn from enough samples in five separate classes.
Therefore, in order to improve the detection accuracy, a YOLOv4-tiny neural network was used. The model was trained using a more versatile and inclusive public dataset. The detection accuracy measured by mAP was improved from 35.9% to 95.6% to detect six classes of vehicles.
The model was trained in the Darknet framework using Google Colab. To implement the model in an FPGA board, we used a PYNQ-Z2 board from Xilinx that supports python coding. The board supports the Jupyter notebook environment. After installing all the required dependencies and libraries that support deep learning and computer vision, such as opencv-contrib and dnn, the darknet model was used for inference from the board using the python notebook. The model runs detection smoothly on testing frames; however, the frame rate was slow when a video was tested. Further investigation in future work can be done to improve the frame rate in a video sequence.

The research and design exploration in this work concluded the possibility of implementing deep learning in FPGA boards toward smart traffic lights control systems. These boards are affordable and can be directly connected to a camera and to a traffic light system, which eliminates the need for a personal computer to run the object detector. This work can also play a key role in improving mobility and reducing waiting time as well as the pollution caused by emissions from cars stuck in traffic jams.

Bibliography

ADMIN, 2021. *Microsoft uses Intel FPGA technology to achieve smarter Bing Search* Available from: https://ee-paper.com/microsoft-uses-intel-fpga-technology-to-achieve-smarter-bing-search/#:~:text=Microsoft%20is%20using%20FPGA%20%28field%20programmable%20gate%20array%29,for%20the%20performance%20of%20Bing%E2%80%99s%20intelligent%20search%20function.

ADRIAN ROSEBROCK, 2017. *Deep Learning with OpenCV* Available from: https://pyimagesearch.com/2017/08/21/deep-learning-with-opencv/

ALBERTO FERNANDEZ VILLAN, 2019. *Mastering OpenCV 4 with Python.* Packt Publishing

ALEXEYAB, *Alexey repo* Available from: https://github.com/AlexeyAB

ANDREW ELBERT WILSON, 2020. *Dynamic Reconfigurable Real-Time Video Processing Pipelines on*
SRAM-based FPGAs, Brigham Young University

ARYAOMNITALK, 2020. *Advanced*
Traffic Management
System - ATMS Available from: https://aryaomnitalk.com/advanced-traffic-management-system-atms/

BAO, C. *et al.,* 2020. A power-efficient optimizing framework FPGA accelerator based on winograd for YOLO. *IEEE Access,* 8, 94307-94317

BESTAMI, G., B.O. SEFA and S.B. HASAN, 2022. LPYOLO: Low Precision YOLO for Face Detection on FPGA.

BEYELER, M., 2017. *Machine Learning for OpenCV.* Birmingham: Packt Ltd

BYUNG-GIL HAN *et al.,* 2020. Design of a Scalable and Fast YOLO for Edge-Computing Devices.

CAMBRIDGE UNIVERSITY, 2005. *Traffic pollution - measuring the real damage* Available from: https://phys.org/news/2005-09-traffic-pollution-real.html

CHANDAN, G., *et al.*, 2018. *Real Time Object Detection and Tracking Using Deep Learning and OpenCV* Available from: https://www.researchgate.net/profile/Ayush-Jain-23/publication/331421347_Real_Time_Object_Detection_and_Tracking_Using_Deep_Learning_and_OpenCV/links/5d70a32f299bf1cb8088576c/Real-Time-Object-Detection-and-Tracking-Using-Deep-Learning-and-OpenCV.pdf

D. GOSHORN *et al.*, 2010. Field Programmable Gate Array Implementation of Parts-Based Object Detection for Real Time Video Applications. *- 2010 International Conference on Field Programmable Logic and Applications.* pp.582-587

D. T. NGUYEN, *et al.*, 2019. *A High-Throughput and Power-Efficient FPGA Implementation of YOLO CNN for Object Detection.* , pp.1861-1873

DARKNET, *YOLO: Real-Time Object Detection*

DAVID, 2022. *keras-YOLOv3-model-set* Available from: https://github.com/david8862/keras-YOLOv3-model-set

DIGANTA MISRA, 2019. *Mish: A Self Regularized Non-Monotonic Activation Function* . Available from: https://arxiv.org/abs/1908.08681

E. RZAEV, A. KHANAEV and A. AMERIKANOV, 2021. Neural Network for Real-Time Object Detection on FPGA. *- 2021 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM).* pp.719-723

EDWARD ZHANG, 2020. *Car Object Detection* Available from: https://www.kaggle.com/datasets/sshikamaru/car-object-detection

EDWARD, R., K. ANTON and A. ALEKSANDR, May, 1 2021. Neural Network for Real-Time Object Detection on FPGA. *Conference: 2021 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM).*

EMMERT-STREIB, F. *et al.*, 2020. An introductory review of deep learning for prediction models with big data. *Frontiers in Artificial Intelligence,* 3, 4

G. CHANDAN *et al.*, 2018. Real Time Object Detection and Tracking Using Deep Learning and OpenCV. *- 2018 International Conference on Inventive Research in Computing Applications (ICIRCA).* pp.1305-1308

G. HARRIAT CHRISTA *et al.*, 2021. CNN-based Mask Detection System Using OpenCV and MobileNetV2. *- 2021 3rd International Conference on Signal Processing and Communication (ICPSC).* pp.115-119

GEORGE MILEV, ASTLEY HASTINGS and AMIN AL-HABAIBEH, 2019. Investigating The Effect of Expanding The Use of Electric Cars On The Environment - A Case Study From Scotland.

GOOGLECOLAB, *Welcome To Colaboratory* Available from: https://colab.research.google.com/#scrollTo=Nma_JWh-W-IF

HENRIK, J., 2015. Evaluating Vivado High-Level Synthesis on OpenCV Functions for the Zynq-7000 FPGA. www.diva-portal.org,

IAN KUON, R.TESSIER and JONATHAN ROSE, 2008. *FPGA Architecture* Available from: https://www.semanticscholar.org/paper/FPGA-Architecture-Kuon-Tessier/921c9c904a79f91b85a4b086cb446c4d33911c3c

J. P. KNIGHT, *J. P. Knight* Available from: https://en.wikipedia.org/wiki/J._P._Knight

JUN SANG *et al.*, 2018. An improved YOLOv2 for vehicle detection.

JUNIPER RESEARCH, 2022. *SMART TRAFFIC MANAGEMENT SYSTEMS TO SAVE 205 MILLION METRIC TONS OF CO2 BY 2027; DRIVEN BY CONGESTION REDUCTION* Available from: https://www.juniperresearch.com/pressreleases/smart-traffic-management-systems-to-save-205m

J. GONG *et al.,* 2020. Vehicle detection in thermal images with an improved yolov3-tiny. - *2020 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS).* pp.253-256

JIANXIAO, Z. *et al.,* Dec 23. MME-YOLO: Multi-Sensor Multi-Level Enhanced YOLO for Robust Vehicle Detection in Traffic Surveillance.

LONGZHEN YU *et al.*, 2022. An Efficient YOLO Algorithm with an Attention Mechanism for Vision-Based Defect Inspection Deployed on FPGA.

LOUISE CROCKETT *et al.*, 2019. *Exploring Zynq MPSoC: With PYNQ and Machine Learning Applications.* Glasgow:

M. A. HEDEYA, A. H. EID and R. F. ABDEL-KADER, 2020. *A Super-Learner Ensemble of Deep Networks for Vehicle-Type Classification.* , pp.98266-98280

M. P. PHILIPSEN *et al.,* 2015. Traffic Light Detection: A Learning Algorithm and Evaluations on Challenging Dataset. - *2015 IEEE 18th International Conference on Intelligent Transportation Systems.* pp.2341-2345

M. DHOUIBI, A. K. BEN SALEM and S. B. SAOUD, 2020. CNN for object recognition implementation on FPGA using PYNQ framework. - *2020 IEEE Eighth International Conference on Communications and Networking (ComNet).* pp.1-6

MAHASHREVETA CHOUDHARY, 2019. *What is Intelligent Transport System and how it works?*Available from: https://www.geospatialworld.net/blogs/what-is-intelligent-transport-system-and-how-it-works/

MARCO WINZKER, 2022. *FPGA VISION REMOTE LAB* Available from: https://www.h-brs.de/de/fpga-vision-lab

MAX N. ROECKER *et al.*, 2018. Automatic Vehicle type Classification with Convolutional Neural Networks.

MARYAM BONEH, 2022. VehicleDetection Available from: https://github.com/MaryamBoneh/Vehicle-Detection/blob/main/README.md

NAEEM AYOUB, 2021. Real-Time On-Board Deep Learning Fault Detection for Autonomous UAV Inspections. https://www.mdpi.com/journal/electronics,

NAGRATH, P. *et al.*, 2021. SSDMNV2: A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2. *Sustainable Cities and Society,* 66, 102692

NHTSA, *NATIONAL HIGHWAY TRAFFIC SAFETY ADMINISTRATION* Available from: https://www.nhtsa.gov/

OPENCV, 2022a. *Deep Neural Networks (dnn module)* Available from: https://docs.opencv.org/4.x/d2/d58/tutorial_table_of_content_dnn.html

OPENCV, 2022b. *OpenCV* Available from: https://opencv.org/about/

PAVEL LAPTEV *et al.*, 2022. Neural Network-Based Price Tag Data Analysis.

PYNQ, 2018. *Base Overlay* Available from: https://pynq.readthedocs.io/en/v2.3/pynq_overlays/pynqz2/pynqz2_base_overlay.html

PYNQ, 2018. *Jupyter Notebooks* [viewed 10/08/ 2022]. Available from: https://pynq.readthedocs.io/en/v2.3/jupyter_notebooks.html

REDMON, J. and A. FARHADI, 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767,*

R. V. KSHIRSAGAR and V. V. DABAHDE, 2015. FPGA-Based Intelligent Traffic Light Controller System Design. 2(4),

REKOR, 2022. *Missouri DOT Selects Rekor Systems' Advanced AI-Driven Solutions to Improve Highway Congestion and Reduce Traffic Fatalities* Available from: https://www.rekor.ai/post/missouri-dot-selects-rekor-systems-advanced-ai-driven-solutions-to-improve-highway-congestion-and-reduce-traffic-fatalities

S. OH, J. -H. YOU and Y. -K. KIM, 2020. Implementation of Compressed YOLOv3-tiny on FPGA-SoC. - *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia).* pp.1-4

S. RUJIKIETGUMJORN and N. WATCHARAPINCHAI, 2017. Vehicle detection with sub-class training using R-CNN for the UA-DETRAC benchmark. - *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS).* pp.1-5

S.L.QADDORI and N.T.GADAWE, 2020. *Real-Time Traffic Light Controller System based on FPGA and Arduino* Available from: https://eudl.eu/pdf/10.4108/eai.28-6-2020.2297938

S.VAMSHI KRISHNA and P.GIRI PRASAD, 2017. Design and Implementation of ITLC System Using FPGA. 12(05),

SMITH, S.F. *et al.*, 2013. Surtrac: Scalable urban traffic control.

SWARCO, *Advanced Traffic Management Systems (ATMS)* Available from: https://www.swarco.com/solutions/traffic-management/highway-and-tunnel/atms#:~:text=The%20Integrated%20Approach%20to%20Traffic,user%20and%20the%20highway%20operator.&text=driving%20conditions%20while%20reducing%20emissions.

T. FANG, X. HUANG and J. SANIIE, 2021. Design Flow for Real-Time Face Mask Detection Using PYNQ System-on-Chip Platform. - *2021 IEEE International Conference on Electro Information Technology (EIT).* pp.1-5

TAL KREISLER , and URIEL KATZ, *No Traffic* Available from: https://notraffic.tech/

TEXAS A&M TRANSPORTATION INSTITUTE, *2021 Urban Mobility Report* Available from: https://mobility.tamu.edu/umr/

THE GUARDIAN, 2005. *Darling unveils road charging plans* Available from: https://www.theguardian.com/politics/2005/jun/09/immigrationpolicy.transport

TRAFIKSOL, 2018. *ADVANCED TRAFFIC MANAGEMENT SYSTEM* Available from: https://www.trafiksol.com/advanced-traffic-management-system/

UNITED NATIONS, 2018. *68% of the world population projected to live in urban areas by 2050* Available from: https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html

V. SZE, *et al.*, 2017. *Efficient Processing of Deep Neural Networks: A Tutorial and Survey.* , pp.2295-2329

VITO ALBINO, UMBERTO BERARDI and ROSA MARIA DANGELICO, 2015. Smart Cities: Definitions, Dimensions, Performance, and Initiatives. 22(1),

WEFORUM, 2018. *Traffic congestion cost the US economy nearly $87 billion in 2018* Available from: https://www.weforum.org/agenda/2019/03/traffic-congestion-cost-the-us-economy-nearly-87-billion-in-2018/

XILINX, 2022a. *How AXI4-Stream Works* Available from: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/How-AXI4-Stream-Works

XILINX, 2022b. *TUL PYNQ™-Z2 board* Available from: https://www.tulembedded.com/FPGA/ProductsPYNQ-Z2.html

X. LI*, et al.*, 2019. *The ParallelEye Dataset: A Large Collection of Virtual Images for Traffic Vision Research.* , pp.2072-2084

Y. LECUN and Y. BENGIO, 1995. Convolutional networks for
images, speech, and time series. In: The handbook of brain
theory and neural networks 3361.10 (1995).

YANN LECUN, YOSHUA BENGIO and GEOFFREY HINTON, 2015. Deep Learning.

Y. LECUN *et al.*, 1989. Backpropagation Applied to Handwritten Zip Code Recognition. Volume 1(Issue 4),

Y. TU *et al.*, 2019. A Power Efficient Neural Network Implementation on Heterogeneous FPGA and GPU Devices. *- 2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI).* pp.193-199

Z. CHEN *et al.*, 2019. Real Time Object Detection, Tracking, and Distance and Motion Estimation based on Deep Learning: Application to Smart Mobility. *- 2019 Eighth International Conference on Emerging Security Technologies (EST).* pp.1-6

Z. DONG, *et al.*, 2015a. *Vehicle Type Classification Using a Semisupervised Convolutional Neural Network.* , pp.2247-2256

Z. HUO, Y. XIA and B. ZHANG, 2016. Vehicle type classification and attribute prediction using multi-task RCNN. *- 2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI).* pp.564-569

Weng, Lilian (2017). “Object detection for dummies part 3: R-CNN family”. In: lilianweng. github. io/lil-log.

Zou, Zhengxia et al. (2019). “Object detection in 20 years: A survey”. In: arXiv preprint arXiv:1905.05055.